USING STATISTICAL ANALYSIS TO GENERATE EXCEPTION RULES THAT ALLOW LEGITIMATE MESSAGES TO PASS THROUGH APPLICATION PROXIES AND GATEWAYS

INVENTORS
ABHISHEK CHAUHAN
RAJIV MIRANI
PRINCE KOHLI

BACKGROUND

Field of Invention

[0001]   The present invention relates generally to web application security

systems, and more particularly, to using application behavior to generate exception rules

to allow legitimate messages to pass through application proxies and gateways.

Background of the Invention

[0002]   Corporations are rapidly deploying web-based applications to automate

business processes and to facilitate real-time interaction with customers, business

partners and employees. Highly vulnerable to malicious hackers, web applications

provide an entry point through which sensitive data can be accessed and stolen. Given

the vulnerability of web applications, establishing a web application protection is critical

for any enterprise that is exposing sensitive data or transaction systems over the Internet.

[0003]   Firewalls are an essential component in a corporate entity's network

security plan. They represent a security enforcement point that separates a trusted

network from an untrusted network. Firewalls determine which traffic should be allowed

and which traffic should be disallowed based on a predetermined security policy.

[0004]   Firewall systems designed to protect web applications are known. They are commonly implemented as application proxies or application gateways. An application proxy is an application program that runs on a firewall system between two networks and acts as an intermediary between a web client and a web server. When client requests are received at the firewall, the final server destination address is determined by the application proxy software. The application proxy translates the address, performs additional access control checking, and connects to the server on behalf of the client. An application proxy authenticates users and determines whether user requests are legitimate.

[0005]   Two approaches for implementing application firewalls are known - a positive security model, which enforces positive behavior; and a negative security model, which blocks recognized attacks.

[0006]   A negative security model blocks recognized attacks by relying on a database of expected attack signatures. This model operates similar to an anti-virus software and is referred to as a "blacklist" solution. All the incoming requests are compared against the known signatures. If a known attack is identified, the request is blocked. One downside of this approach is that a signature list needs to be always kept up-to-date. Otherwise, when a new request comes in that is not on the signature list, it will be passed to the server for processing. As a result, illegitimate traffic can get through.

[0007]   A positive security model enforces positive behavior by learning the application logic and then building a security policy of valid known requests as a user interacts with the application. This approach explores all possible entry points, such as URLs, in a web application, and generates a set of allowable user actions for a particular

-2-

application program. Messages that match some of the allowable actions are allowed. The rest of the messages are blocked. This solution is commonly referred to as a "whitelist" technique. One problem with this approach is that the generated list of allowable actions is static. As a result, when the application program changes, some of the rules become obsolete. In addition, certain situations, such as extensive use of JavaScript or deliberate deviations of the application from its usual behavior can generate a large number of false positives, i.e., illegitimately blocked messages. Since some of the blocked messages may represent legitimate requests rather than malicious attacks, the existing model blocks messages which may turn out to be legitimate.

[0008]   Accordingly, what is needed is a web application security technique that generates exception rules to the rules that block legitimate traffic.

SUMMARY OF THE INVENTION

[0009]   A security gateway receives messages rejected by a message filter based on a set of rules. The security gateway extracts attributes of the rejected messages that triggered the rules. The security gateway maintains frequencies with which the messages with a particular attribute were rejected by the rules. The security gateway finds rejected messages or attributes having a high frequency of occurrence. Since messages or attributes having a high frequency of occurrences are more likely to represent legitimate requests rather than malicious attacks, the security gateway generates exception rules, which would allow messages that have similar attributes to pass through the gateway.

[0010]   In one embodiment, initially, messages or components thereof are received by a message filter. Examples of the received messages are URL requests, forms, and other client-server communications. The message filter extracts attributes from the messages. The attributes may be data from the message, or derived values from

-3-

22271/08770/SF/5114680.4

the data, such as statistical information. Some messages will be rejected because they trigger at least one rejection rule used by the message filter. Exemplary rules used by the message filter are: an immutable property for a given cookie cannot be modified, a form field cannot exceed a certain number of characters, a URL that is not on the list of allowed URLs can not pass.

[0011]   An adaptive filter receives rejected messages along with an indication of the rule (or rules) that has caused a rejection and an attribute (or attributes) of the message that triggered the rule. The adaptive filter generates a rejection list that includes a message, rule, and attribute of the message that triggerred the rejection and passes the rejection list to a learning engine.

[0012]   A frequency is maintained to indicate a number of occurrences a message with a particular attribute was rejected by a rule. These frequencies create a distribution of different rule violations in terms of various message attributes. The learning engine determines which frequencies exceed a threshold and generates an exception rule to the message filter's rejection rule which rejected the message containing the attribute. Exception rules thus describe attributes with high frequency occurrences. Exception rules are provided to the adaptive filter, which allows messages to pass according to the exception rules.

[0013]   Alternatively, the functionality of the adaptive filter is embedded into the message filter. The generated exception rules are provided to the message filter. The message filter is configured to traverse to the exception rules directly from the filtering rules. The message filter allows messages to pass through the security gateway according to the exception rules.

[0014]   The security gateway is adapted to generate exception rules for rejected URL messages according to another embodiment. In this embodiment, all rejected URLs are stored in a trie structure. Each node in the trie structure is associated with a component of a URL message. A frequency count is maintained at each node to indicate a number of occurrences of URLs which were rejected. The learning engine traverses the trie structure to select a node (or nodes) to generate a cut. A cut is a collection of nodes for which exception rules will be generated. Nodes that are selected for the cut meet a set of constraints with respect the frequency information. The learning engine generates an exception rule that will allow a URL associated with the selected node and its descendants to pass.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015]   Fig. 1 is a block diagram of environment in which the invention operates;

[0016]   Fig. 2 is a block diagram of an embodiment of a system for adaptively filtering messages;

[0017]   Fig. 3 is a flow chart of a method performed by the security gateway in one embodiment of the present invention;

[0018]   Fig. 4 is a trie structure for storing rejected URLs in another embodiment of the present invention; and

[0019]   Fig. 5 is a flow chart of a method performed by the security gateway according to another embodiment of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS
1. System Architecture Overview

[0020]   Fig. 1 is a block diagram of an environment 100 in which the invention operates. The environment 100 includes a client system 110 and a server system 120

-5-

connected by a communication network 150. A security gateway 130 is interposed

between and connected to client 110 and server 120. Although this invention is described

in the context of the client-server architecture, it should be understood that the invention

can be implemented in any other suitable architecture, such as a peer-to-peer architecture

where each system 110 and 120 can be a client and a server.

[0021] As used herein, the term "server system" 120 simply denotes those aspects

of the software program associated with a remote computer, as well as underlying

operating system and hardware support. As will be understood by those of skill in the art,

a server system 120 within the context of the present invention can comprise components

of the software program, as well as components of the operating system of a remote

computer and hardware components of a remote computer. Server system 120 may

include a web infrastructure, such as a web server 140, an application server 160, and a

database 170. Web server 140 is a computer running software for serving content across

the Internet, such as for example Microsoft Internet Information Server (IIS), or Apache.

Web server 140 responds to requests received from client system 110 by transmitting

HTML pages across the Internet for display by a web browser (not shown) that is running

on client system 110. Application server 160 is a program that handles all application

operations between users an enterprise's backend business applications or databases.

Database 170, in turn, stores all data relevant to the enterprises' operations. The server

system 120 may comprise one or more computers for implementing the above described

functionality and services.

[0022] As used herein, the term client system 110 simply denotes those aspects of

the software program associated with a user's computer, as well as underlying operating

system and hardware support. As will be understood by those of skill in the art, a client

-6-

system 110 within the context of the present invention can comprise components of the software program, as well as components of the operating system of a user's computer and hardware components of a user's computer. Client system 110 includes a web browsing functionality. While FIG. 1 illustrates a single client system 110, it is appreciated that in actual practice there will be any number of client systems 110 that communicate with the server system 120.

[0023]   In one implementation, security gateway 130 operates as a proxy in that it is installed directly in front of server 120. In yet another implementation, security gateway 130 operates as a gateway between the public Internet and an internal network (not shown), such as a WAN, or LAN. This implementation ensures that all incoming and outgoing messages are inspected for compliance with defined security policies so that all of the network's assets can be protected against new, yet-undefined threats. Network administrators can validate application security policies before deployment.

[0024]   Communication network 150 can be the Internet, and particularly, the World Wide Web portion thereof. However, it will also be appreciated that communication network 150 can be any known communication network.

[0025]   Security gateway 130 rejects messages based on a set of rules. Security gateway 130 gathers statistical data to determine the frequencies with which messages with a particular attribute were rejected. Security gateway 130 compares the frequencies to a threshold. Security gateway 130 generates exception rules for high frequency messages or attributes thereof, which rules would allow messages that have similar attributes to pass through security gateway 130 according to the exceptions.

[0026]   Fig. 2 is a block diagram of functional modules of security gateway 130. These modules include a message filter 210, an adaptive filter 220, and a learning engine

-7-

230. As used herein, the term "module" refers to computer program logic and/or any hardware to provide the functionality attributed to the module.

[0027] Message filter 210 is adapted to perform filtering of data as is well-known in the art. Message filter 210 maintains a list of filtering rules, also referred as "rejection rules." Message filter 210 processes rules in a list form, checking messages against the rules in order, until the message is either accepted or rejected. If no rule applies, then the default policy (either reject or accept) is applied. Message filter 210 can be any known product, such as the one developed by Check Point™ Software Technologies, Ltd, of Redwood City, California. Message filter 210 passes rejected messages to adaptive filter 220 along with an indication of the rule (or rules) that has caused a rejection and an attribute of the rejected message that triggered the rule.

[0028] Adaptive filter 220 receives the rejected message and by default rejects the messages. Adaptive filter 220 generates a rejection list that includes a rejected message, the rule (or rules) that has caused the rejection and an attribute of the rejected message that triggered the rule. Adaptive filter 220 is further adapted to receive exception rules generated by a learning engine 230 (as will be discussed in more detail below) and to apply the exception rules to allow some of the rejected messages.

[0029] The learning engine 230 is adapted to receive the rejection list. For each attribute on the rejection list, learning engine 230 maintains a frequency (also referred here is as a "frequency count" or a "count") indicating a number of occurrences of a message with the identified attribute that was rejected by a rule. Learning engine 230 determines whether the frequency count of a message with the identified attribute exceeds a threshold. If the frequency count exceeds the threshold, the learning engine 230 generates exception rules to the rules which would otherwise reject messages with

-8-

this attribute. Generated exception rules are for high frequency attributes. Since messages with high frequency attributes are not likely to be malicious attacks, but rather more likely to represent legitimate requests, the exception rule allows these legitimate requests to pass through the gateway 130. For example, when a large number of users request a particular URL that is not on the list of allowable URLs, this may indicate that the request is more likely to be a legitimate one, rather than a malicious attack.

[0030] In another embodiment, the functionality of adaptive filter 220 is embedded into message filter 210. The exception rules are provided to message filter 210, which is configured to traverse to the exception rule directly from the filtering (rejection) rule. Message filter 210 allows messages to pass through security gateway 130 according to the exception rules.

[0031] Memory 250 stores a number of data structures. Memory 250 maintains a rejection list generated by adaptive filter 220. Memory 250 also stores various thresholds, frequency counts, and exception rules.

2. Methods of Operation

[0032] Referring now to FIG. 3 there is a shown a flowchart of a method of operation in accordance with one embodiment of the invention. Initially, at step 305, message filter 210 receives messages and components thereof and filters the messages using filtering rules. Examples of the received messages are a web page, a URL, and other client-server communications. Examples of the received message components are cookies, form fields, and hidden fields. Messages or components thereof are checked against the rules in order, until the messages are either accepted or rejected. If a message triggers a rejection rule, the message is rejected. If no rule applies, then the default

policy (either reject or accept) is applied. Exemplary data filtering rules are shown below in Table 1.

Table 1: Filtering Rules

Rule No.	Rule
1	Cookies' unique ID fields cannot be tampered
2	No missing cookie
3	No extra cookie
4	Hidden field cannot be of a bigger length than specified
5	Password field cannot exceed 5 characters and user name field cannot exceed 20 characters
6	Do not allow URLs that are not on a list of allowable URLs

[0033] At this point it is beneficial to explain the rules listed above in Table 1.

Cookies (Rules 1, 2, and 3)

[0034] A cookie is a piece of data given to a web browser running on client 110 by server 120. A session cookie is a cookie that is assigned to a user by server 120 on a per session basis and it is stored on client 110 only for the user session. Session cookies will typically store information in the form of a session identification and may contain related state information for the session. For example, server 120 may issue the following command: set cookie: session ID=12345678. The browser stores the cookie in a file. The value of the cookie is stored on server 120. The cookie is sent back to server 120 each time the web browser requests a web page from the server 120. Message filter 210 checks the cookie against the filtering rules in order. If one of the rules indicates that the cookie session ID field cannot be changed, message filter 210 accesses server 120 to compare the returned cookie value with the stored value. If the returned value is the same as the stored value, the message is allowed to pass (if the message does not trigger

-10-

any other rule). If the value is different, it triggers rule 1 and the message is rejected. Server 120 stores the new value of the cookie. Message filter 210 forwards the rejected message along with an indication of the rule (or rules) that rejected the message and an attribute of the message that triggered the rule. It should be understood that an attribute can be a message itself, a component of the message, a value of some components of the message itself, or a derived value, such as a data type, a length, count, or any other property, statistical characteristic or the like. Thus, in this example, the attribute is a component of the message, i.e., session ID=12345678.

[0035] Similarly, when server 120 sends to a web browser a cookie (or cookies), the web browser has to return to server 120 the same number of cookies. If one of the rules indicates that there should not be any missing cookie, message filter 210 checks the number of cookies sent by the web browser with the number of cookies required by server 120. If the number of cookies returned by the web browser less than the number of cookies required by the server 120, rule 2 is triggered and the message gets rejected with an attribute of the message indicating which cookie is missing. Similarly, if the message triggers rule 3 (it includes an extra cookie) the message is rejected with the attribute indicating the extra cookie.

Hidden Fields and Form Fields (Rules 4 and 5)

[0036] Web pages have hidden fields and form fields. A rule may require that a password field or a user login field on a web page cannot exceed a certain number of characters. Each message is checked against filtering rules. If the number of characters provided by a user exceeds a certain number of characters limited by a rule, the rule is triggered. As a result, the message is rejected along with an attribute of the message that triggered the rule, i.e., the length of the field that triggered the rule.

-11-

URLs (Rule 6)

[0037]   In one implementation, message filter 210 stores a list of all URLs that can be requested by a user. All incoming URL requests are checked against this list. If the request is made for a URL that is not on the list, then the request is rejected along with an attribute of the message that triggered the rule. In this case, the attribute is a message itself, i.e., the rejected URL.

[0038]   It should be noted that security gateway 130 can operate in a learning mode, in a blocking mode, and both in the learning mode and the blocking mode concurrently. In the learning mode, security gateway 130 receives rejected messages, maintains statistics for rejected messages and generates exception rules, which would allow rejected messages to pass. In the blocking mode, security gateway 130 allows rejected messages to pass according to exception rules. When security gateway 130 operates both in the learning mode and the blocking mode concurrently, it checks for an applicable exception rule for the rejected message. If no exception rule is found, security gateway 130 adds the message to the list of rejected messages and updates statistics. These modes of operation are discussed in more detail below.

[0039]   Referring again to Fig. 3, adaptive filter 220 receives 310 a rejected message along with an indication of the rule (or rules) that rejected the message and an attribute (attributes) of the message that triggered the rule. Adaptive filter 220 updates a rejection list. The elements of the rejection list are tuples (rejected message, rule, and attribute). Exemplary records stored in the rejection list are shown below in Table 2.

-12-

Table 2. Rejection List

Message	Rule	Attribute
User=John; PW=1234567	5	Length of PW=7 characters
button.x missing	3	Button.x
Get seen_it cookie=extra	2	Seen_it cookie
www.cnn.com/2004/Jan	6	www.cnn.com/2004/Jan
User=JackieJackieJackieJackie	5	Length of User name= 24 characters

[0040] As shown in Table 2, the first message on the rejection list triggered rule 5 because a user entered the password that exceeded the number of characters allowed to be in the password field of a web form. The attribute is the length of the password field, 7 characters.

[0041] The second message triggered rule 3 because cookie button.x was missing when a user sent a new request to server 120. The attribute is the missing button.x cookie.

[0042] The third message ended up on the rejection list because seen_it cookie was an extra cookie in the user request to server 120. The attribute is seen_it cookie.

[0043] The fourth message was rejected by rule 6 because it was a request for a URL that is not on the list of allowed URLs. The attribute is rejected URL www.cnn.com/2004/Jan.

[0044] Finally, the fifth message was rejected because the user name exceeded 20 characters. The attribute is the length of the user name field, 24 characters.

[0045]   Adaptive filter 220 forwards the rejection list to learning engine 230.

Learning engine 230 maintains a frequency count to indicate a number of occurrences a

message with a particular attribute was rejected by a rule.

Table 3: Attributes' Frequency Counts

Rule	Attribute	Count
5	Length of PW=7 characters	150
3	Button.x	15
2	Seen_it cookie	120
6	URL www.cnn.com/2004/ Jan	50
5	Length of User name= 24 characters	160

[0046]   These frequencies create a distribution of different rule violations in terms

of various message attributes.

[0047]   It should be noted that security gateway 130 analyzes messages based on a

user session. A user session is a unit of activity that a unique user spends on a web site

during a specified period of time. The number of user sessions on a web site is used in

measuring the amount of traffic on a particular web site. A web site administrator

determines the time frame of a user session (e.g., 30 minutes). If a user comes back to

the site within that time period, it is still considered one user session because any number

of visits within that time frame will only count as one session. If the visitor returns to the

site after the allotted time period has expired, for example, an hour from the initial visit,

then it is counted as a separate user session. Accordingly, in one implementation, if more

than one message is rejected with the same attribute during a particular user session, the

count for that attribute is incremented only once. This type of count is referred to as a

-14-

"direct" count. For example, if three messages were rejected during one user session because seen_it cookie was an extra cookie, learning engine 230 increments 320 the count for an attribute seen_it cookie only once. This approach is beneficial because it avoids counting duplicate attributes for rejected messages associated with the same user.

[0048] Frequency counts can be weighted. A weighted count is incremented depending on other attributes of the message, such as its source. For example, if the message came from a trusted IP address, the weighted count is incremented, for example, by 100. If the message came from a non-trusted IP address, the weighted count is incremented, for example, by 1.

[0049] Maintaining frequency counts for each attribute is beneficial because it allows security gateway 130 to examine the patterns of various message rejections. The frequencies are used to generate exception rules that would allow messages with high frequency attributes to pass through security gateway 130.

[0050] The learning engine 230 determines 325 which frequency counts exceed a threshold. At this point, it is useful to define how thresholds are calculated. In one implementation, memory 250 stores a sensitivity parameter specified by a system administrator. A system administrator can change the sensitivity parameter via a user interface (not shown).

- Let sensitivity parameter S be a fraction of the total number of messages that should be passed by security gateway 130, such as 25%;

- Let L be a total number of requests over a predetermined time interval, e.g., 500 messages per minute;

- Threshold T is a product of S and L. T=25% x 500=125.

-15-

[0051]   If the frequency count of the attribute exceeds the threshold, learning

engine 230 generates 330 an exception rule to the rule which would otherwise reject a

message with that attribute.

[0052]   Referring again to Table 3, it is shown that frequency counts for the first,

third and fifth attribute in the table exceed the threshold of 125. Learning engine 230

generates 330 exception rules to the rules that rejected messages with these attributes.

Exception rules thus describe attributes with high frequency occurrences. A list of

generated exception rules is shown below in Table 4. As shown in Table 4, the exception

rules are sorted by the rejection rule number.

Table 4. Exception Rules

Rule	Exception
2	seen_it cookie
5	Length of PW=7 characters
5	Length of User name= 24 characters

[0053]   As shown in Table 4, for any given rule, more than one exception rule can

be generated. Adaptive filter 220 uses generated exception rules to allow rejected

messages, as will be described in more detail below.

[0054]   Referring again to Fig. 3, once exception rules have been generated for

high frequency attributes, adaptive filter 220 may allow some messages to pass according

to the exception rules. More particularly, in one implementation, once adaptive filter 220

received a rejected message along with an indication of the rule (or rules) that rejected

the message and an attribute (or attributes) of the message that caused the rejection.

Adaptive filter 220 traverses the list of all the exception rules generated for the rejection

rule. Adaptive filter 220 uses the rejection rule number to index or hash into the list of

-16-

exception rules. If adaptive filter 220 finds an exception rule that matches the attribute of the rejected message, it allows 340 the message to pass. If the matching exception rule is not found, adaptive filter 220 adds the message to the rejection list and updates the frequency count for the attribute.

[0055] To continue with the same example, rule 2 rejected a new message with an attribute: seen_it cookie. Adaptive filter 220 indexes into Table 4 to find all the exception rules generated for rule 2. Adaptive filter 220 then finds the exception rule that matches the attribute of the rejected message (seen_it cookie). Adaptive filter 220 allows the message to pass according to the exception rule.

[0056] Alternatively, a list of generated exception rules is provided to message filter 210. Message filter 210 is configured to process rules in a list form, check the messages agains the rejection rules, and traverse to the exception rules directly from the filtering rules. If message filter 220 finds an exception rule that matches a rejected message or an attribute thereof, message filter 210 allows the message to pass through security gateway 130 according to the exception rule.

Alternative Embodiment

[0057] This embodiment addresses an alternative method of generating exception rules to the rules that reject URL messages.

[0058] At this point it will be useful to explain a structure of a typical URL. Assume that the following URLs were rejected by message filter 210:

http://www.cnn.com/2004/Jan/1/sports.html

http://www.cnn.com/2003/Jan/ and

http://www.cnn.com/2002/Dec

-17-

[0059]　Each URL string is a combination of the protocol, such as "HTTP://", a host domain name, such as "www.cnn.com", and a directory path, such as 2004/Jan/1/sports. The directory path has one or more URL components, such as "2004," "Jan," "1," and "sports". The directory path may end with a suffix, such as ".html," which indicates the coded format language used to create hypertext documents. The URL components of a directory path are in a hierarchical relationship with each other. For example, in the URL identified by the path www.cnn.com/2004/Jan/1/sports.html, component "2004" is of a higher order than component "Jan." Similarly, component "Jan" is of a higher order than component "1." Higher order components are in an ancestor-descendants relationship with lower order components. For example, component "2004" is a parent of component "Jan" and component "Jan" is a child of component "2004."

[0060]　In one implementation, learning engine 230 stores all URLs with the same host domain name in a trie structure. Each time a URL with the same host domain name is rejected, a trie structure is updated with that URL. Using trie structures to represent data is well-known in the art. Nodes in a trie structure can be in an ancestor – descendant relationship. A trie structure has a root node, which is the topmost node that has no ancestor nodes. A trie structure also has leaf nodes and internal nodes. An internal node is a node that has one or more child nodes. A leaf node has no children. All children of the same node are siblings.

[0061]　Each node in a trie structure is associated with a URL component. Fig. 4 shows a trie structure that stores all rejected URLs with the same domain name www.cnn.com. As shown in Fig. 4, www.cnn.com is a root node of the structure. Each node is associated with a URL component. Each node is referenced by a URL path. For

22271/08770/SF/5114680.4

example, nodes "2004," "2003," and "2002" are referenced by URL paths www.cnn.com/2004, www.cnn.com/2003, and www.cnn.com./2002 respectively. Each node "2004," "2003," and "2002" has children, and therefore they are internal nodes. Nodes "sports", "headlines", "Dec", "Jan", and "Feb" are leaf nodes because they have no children.

[0062]   At each node associated with a URL component of a rejected URL, learning engine 230 also maintains frequency counts indicating a number of occurrences of URLs associated with the nodes which were rejected. It should also be noted that the frequency counts are also maintained at those nodes that are associated with the components of the URLs that were not explicitly requested. These frequency counts are determined as a function of a number of occurrences with which a URL component associated with a node and its descendants were rejected by a rule. An example of the function is a linear function. To illustrate this point, consider the following example.

If rejected URLs are

a/b/c

a/b/d, and

a/b,

then the counts on leaf nodes c and d are determined as follows:

the count on node c equals the number of times the URL with the path a/b/c was rejected;

the count on node d equals the number of times the URL with the path a/b/d was rejected;

the count on internal node b is determined as follows:

the count on node b = the number of times URLs that include the URL component "b" were rejected.

[0063]   Thus, if URL a/b/c was rejected 20 times, URL a/b/d was rejected 40 times, and URL a/b was rejected 30 times, the count on b: 20+40+30=90.

[0064]   If URL a/b were not rejected, then the count on node b: 20+40=60.

[0065]   Using the algorithm described above, frequency counts are determined for all the nodes in the trie structure shown in Fig. 4 as follows:

[0066]   The count on the node that includes URL component "1" and referenced by the path www.cnn.com/2004/Jan/1 is determined as follows:

- www.cnn.com/2004/Jan/1 was rejected 30 times (this number is shown in Fig. 4 on the outgoing edge connecting the nodes referenced by the path www.cnn.com/2004/Jan and www.cnn.com/2004/Jan/1)

- www.cnn.com/2004/Jan/1/headlines.com was rejected 20 times

- www.cnn.com/2004/Jan/1/sports.com was rejected 30 times

Accordingly, the total count on the node that includes the URL component "1" and referenced by the path www.cnn.com/2004/Jan/1 = 30+20+30= 80.

[0067]   The count on the node that includes a URL component "Jan" and referenced by the path www.cnn.com/2004/Jan is determined as follows:

- www.cnn.com/2004/Jan/1 was rejected 80 times

- www.cnn.com/2004/Jan/26 was rejected 20 times

- www.cnn.com/2004/Jan/30 was rejected 80 times

Accordingly, the total count on the node referenced by the path www.cnn.com/2004/Jan = 80+20+80=180.

Table 5 below shows frequency counts for some of the nodes in the trie structure of Fig. 4.

Table 5. Frequency Counts for Rejected URLs

Rejected URL	Frequency Count
www.cnn.com/2004/Jan/1/sports.html	30
www.cnn.com/2004/Jan/1/headlines.html	20
www.cnn.com/2004/Jan/30/weather.html	50
www.cnn.com/2004/Jan/1	80
www.cnn.com/2004/Jan/26	20
www.cnn.com/2004/Jan/30	80
www.cnn.com/2004/Jan	180

[0068]   Referring now to Fig. 5, it shows a flow chart of a method for generating exception rules to the rules that reject URLs according to this embodiment. Briefly, the method includes the following steps: traversing a trie structure to select a node (or nodes) for a cut and generating exception rules for the selected nodes and their descendants. Those nodes that are not selected in the cut are "dropped", i.e., no exception rules will be generated for these nodes and their descendants.

1. Selecting nodes for a cut

[0069]   At step 510, learning engine 230 traverses the trie structure to select nodes in a cut. The cut is a collection of selected nodes. No node in the cut has an ancestor-child relationship.

-21-

[0070] Learning engine 230 traverses the trie structure using a depth-first search algorithm. The depth-first search algorithm is a search algorithm that explores the first child node before exploring siblings of that node. In one implementation, to create a cut, learning engine 230 selects the deepest node (furthest from the root node) that meets the following two constraints:

1. the node has a count that exceeds a threshold; and

2. none of the children of the selected node have counts exceeding the threshold.

[0071] The above constraints are representative of one known implementation of an algorithm for creating a cut. It should be noted that there are different variations of constraints used to select nodes in the cut. For example, in another implementation, learning engine 230 selects a node with a count exceeding a threshold, without considering the counts of the child nodes.

[0072] As described above, in one implementation, the threshold is determined as a product of a sensitivity parameter and the total number of the messages over a certain time interval.

- Let sensitivity parameter S be a fraction of URL requests that adaptive filter 220 should allow to pass, such as 10%;

- Let L=1000 be a total number of URL requests over a configurable time interval

- Threshold T=SxL=10% x 1000=100

[0073] Continuing with the example shown in Fig. 4, learning engine 230 traverses the trie structure and selects the node referenced by the path www.cnn.com/2004. This node has a count of 240, which exceeds the threshold of 100.

-22-

[0074]   Learning engine 230 continues traversing the structure and selects the child node of that node, which is referenced by the path www.cnn.com/2004/Jan. This node has a count of 180, which exceeds the threshold of 100. Therefore, learning engine 230 continues traversing the trie structure and selects the child node of that node, which is referenced by the path www.cnn.com/2004/Jan/1. Learning engine 230 determines that this node has a count of 80, which is below the threshold. Learning engine 230 continues traversing the trie structure and finds that other children of the node identified by the path www.cnn.com/2004/Jan have a count of 20 and 80 respectively. These counts do not exceed the threshold of 100.

[0075]   Learning engine 230 selects the node identified by the path www.cnn.com/2004/Jan to be in the cut, at step 520, since this node met the two constraints discussed above. It should be noted that once learning engine 230 finds children with the counts below the threshold, learning engine 230 does not need to traverse to other descendants to determine their counts. In this embodiment the count on any node is always equal or greater of the count of the child node. Accordingly, if the count of the child node does not exceed the threshold, then no descendant nodes of the child node can have a count above the threshold.

[0076]   Continuing with the example shown in Fig. 4, learning engine 230 continues traversing the trie structure and selects the node that is identified by the path www.cnn.com/2003. The count on this node is determined as follows: 40+10+15=65. Since the count is below the threshold of 100, this node will not be selected for the cut. Accordingly, no exception rules will be generated for the URL that references this node and its descendants.

[0077]   Learning engine 230 continues traversing the trie structure and identifies the node referenced by the URL path www.cnn.com/2002 with the count of 200. Learning engine 230 determines that this node does not have child nodes with the counts that exceed the threshold. Indeed, child nodes identified by URLs www.cnn.com/2002/Dec, www.cnn.com/2002/Jan, and www.cnn.com/2002/Feb have frequency counts of 50, 70 and 80 respectively. As a result, the node referenced by the URL path www.cnn.com/2002 will be selected for the cut.

2. Exception Rules Generation

[0078]   Although the children of the nodes in the cut (and therefore their descendants) did not individually meet the threshold (i.e., their frequency counts were below the threshold), their ancestors have counts exceeding thresholds. The current approach assumes that if rejected URLs associated with the descendant nodes collectively met the threshold then the rejected URLs were legitimate requests. Accordingly, exception rules are generated to allow these URLs to pass. This serves to generalize the characteristics of the descendant nodes into an appropriate exception rule.

[0079]   For each node in the cut, learning engine 230 generates an exception rule, at step 530, that would allow the URLs associated with that node and its descendants to pass.

[0080]   The exception rule has the format of a typical URL:

<host domain name> <directory path>/suffix.

Directory path includes the following components:

<prefix>/data type 1/data type 2/ .../data type n/.

[0081]   Prefix, which is a path from the host domain name, which is a root node in a trie structure, to the node selected for the cut. In our example, prefix for the first

-24-

selected node is www.cnn.com/2004/Jan/ and prefix for the second selected node is www.cnn.com/2002.

[0082]   The suffix is the common suffix that satisfies all URLs associated with the selected node and its descendants. In our example, suffix .html. satisfies al the URLs associated with the selected node referenced by the path www.cnn.com/2004/Jan and its descendants. No suffix is created for the node identified by the path www.cnn.com/2002 and its descendants.

[0083]   Data type 1 through data type n are components of the directory path. Data type 1 is a data type of a URL component at a level below www.cnn.com/2004/Jan, data type 2 is a data type of a URL component at a level below www.cnn.com/2004/Jan/1 and so forth. Each data type is determined using a data type inferencing algorithm described in a copending application entitled "Inferencing Data Types of Message Components to Generate Rules that Filter Messages Passing Through Security Gateways," which disclosure is incorporated herein by reference.

[0084]   Briefly, the scalar data type inferencing algorithm receives a set of keywords, such as URL components, determines whether each keyword is of a known data type, and finds the most restrictive data type that covers at least T percent of keywords, where T is a configurable threshold. The algorithm creates a Directed Acyclic Graph of the data type set. Each node in the graph is associated with a data type in the data type set. Examples of data types are: Date, Time, INT for Integer, CHAR for character. For each data type, a match factor is computed. The match factor is the percentage of keywords that belong to a particular data type. The most restrictive data type is the data type that meets the following two constraints:

   1. the match factor of the most restrictive data type exceeds T; and

2. the most restrictive data type does not have child nodes with a match factor exceeding T.

[0085]   Referring again to the process of generating exception rules for the selected URLs. Learning engine 230 identifies for each selected node the most restrictive data type that covers descendant nodes of the selected node at a given level. Using the algorithm described above, learning engine 230 determines that "INT" is the most restrictive data type that describes the nodes at one level below than the selected node referenced by the path www.cnn.com/2004/Jan. Data type "CHAR" is the most restrictive data type that describes the nodes at two levels below than the selected node referenced by the path www.cnn.com/2004/Jan. Data type "INT" is the most restrictive data type is that describes the nodes at one level below than the selected node referenced by the path www.cnn.com/2002/. These data types become data types of the components of the generated exception rules.

[0086]   Continuing with the same example shown in Fig. 4, the following exception rules will be generated:

1. www.cnn.com/2004/Jan/<INT>/<CHAR>/html and

2. www.cnn.com/2002/<INT>.

These exception rules will allow the selected nodes referenced by the following URLs: www.cnn.com/2004/Jan and www.cnn.com/2002 as well as their descendant nodes to pass.

[0087]   Thus, although the URLs associated with children (and therefore all descendants) of the nodes selected in the cut have not individually met the threshold, they collectively met the threshold. It should be noted that under the frequency-based algorithm described in the preferred embodiment, no exception rules would have been

-26-

generated to allow the URLs associated with the descendants of the selected nodes to pass, because frequency counts for these URLs are below the threshold.

[0088]   When adaptive filter 220 receives a rejected URL, it finds a matching exception rule for the rejected URL. For example, if the rejected URL www.cnn.com/2004/Jan/12/weather.html, adaptive filter 220 finds the exception rule that has a domain name www.cnn.com, components "2004", "Jan", "12", and "weather," and suffix "html." Adaptive filter 220 finds the matching exception rule that has the same domain name and URL components "2004" and "Jan." Adaptive filter 220 determines the data type of URL component 12, which is INT, and the data type of URL component "weather," which is CHAR. Adaptive filter 220 allows the rejected URL www.cnn.com/Jan/12/weather.html to pass because it matches the exception rule www.cnn.com/Jan/<INT>/<CHAR>/.html.

[0089]   Thus, the security gateway uses statistical analysis to identify legitimate messages and to generate exception rules for legitimate messages, which rules would allow similar messages to pass.

[0090]   The present invention has been described in particular detail with respect to the two possible embodiments. Those of skill in the art will appreciate that the invention may be practiced in other embodiments. First, the particular naming of the components, capitalization of terms, the attributes, data structures, or any other programming or structural aspect is not mandatory or significant, and the mechanisms that implement the invention or its features may have different names, formats, or protocols. Further, the system may be implemented via a combination of hardware and software, as described, or entirely in hardware elements. Also, the particular division of functionality between the various system components described herein is merely

-27-

exemplary, and not mandatory; functions performed by a single system component may instead be performed by multiple components, and functions performed by multiple components may instead performed by a single component.

[0091]   Some portions of above description present the features of the present invention in terms of algorithms and symbolic representations of operations on information. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. These operations, while described functionally or logically, are understood to be implemented by computer programs. Furthermore, it has also proven convenient at times, to refer to these arrangements of operations as modules or by functional names, without loss of generality.

[0092]   Unless specifically stated otherwise as apparent from the above discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0093]   Certain aspects of the present invention include process steps and instructions described herein in the form of an algorithm. It should be noted that the process steps and instructions of the present invention could be embodied in software, firmware or hardware, and when embodied in software, could be downloaded to reside on and be operated from different platforms used by real time network operating systems.

[0094] The present invention also relates to an apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general-purpose computer selectively activated or reconfigured by a computer program stored on a computer readable medium that can be accessed by the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, application specific integrated circuits (ASICs), or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus. Furthermore, the computers referred to in the specification may include a single processor or may be architectures employing multiple processor designs for increased computing capability.

[0095] The algorithms and operations presented herein are not inherently related to any particular computer or other apparatus. Various general-purpose systems may also be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these systems will be apparent to those of skill in the, along with equivalent variations. In addition, the present invention is not described with reference to any particular programming language. It is appreciated that a variety of programming languages may be used to implement the teachings of the present invention as described herein, and any references to specific languages are provided for disclosure of enablement and best mode of the present invention.

[0096] The present invention is well suited to a wide variety of computer network systems over numerous topologies. Within this field, the configuration and management

of large networks comprise storage devices and computers that are communicatively coupled to dissimilar computers and storage devices over a network, such as the Internet.

[0097] Finally, it should be noted that the language used in the specification has been principally selected for readability and instructional purposes, and may not have been selected to delineate or circumscribe the inventive subject matter. Accordingly, the disclosure of the present invention is intended to be illustrative, but not limiting, of the scope of the invention, which is set forth in the following claims.